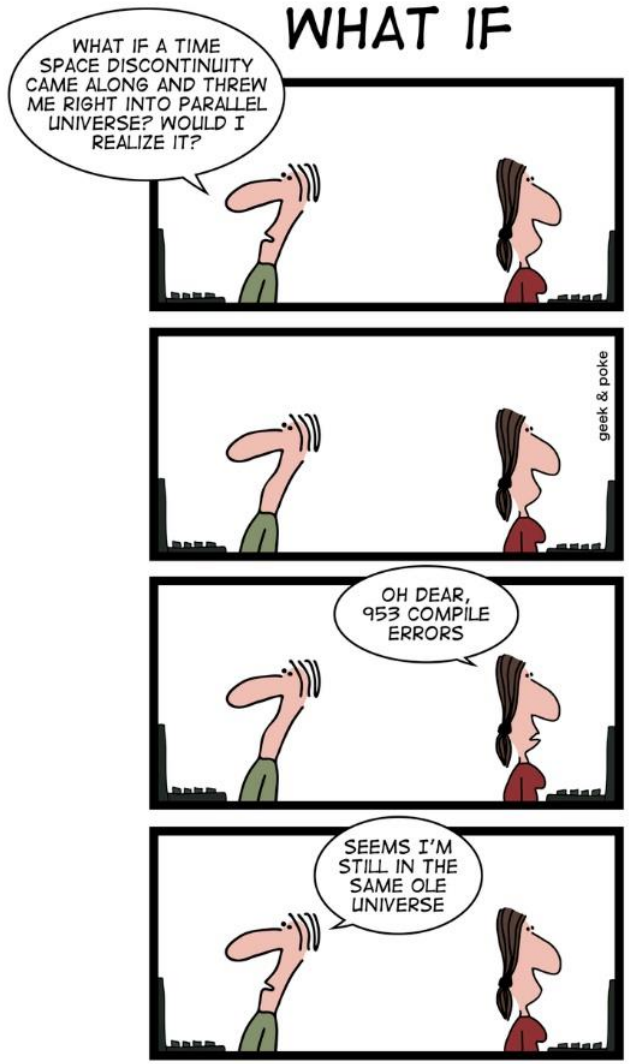
Les conditions

Antoine SCHMID, CFPT-Informatique
Avec la collaboration du CPLN

Table des Matières

- Résumé de l’épisode précédent
- Ajouter des conditions
- Imbriquer des conditions
- Créer des conditions composées
- Exprimer un choix (switch)



Dans l’épisode précédent…

- Interactions
  - prompt
  - alert
- Conversion
  - Number
- Respectez les normes !

Exprimer une condition

- Comment se traduit cet algorithme en JavaScript ?

```
Saisir un nombre
Si ce nombre est positif
  Afficher un message
```

```
const nombre = Number(prompt("Entrez un nombre :"));
if (nombre > 0) {
  console.log(nombre + " est positif");
}
```

if

```
if (condition) {
  // instructions exécutées quand la condition est vraie
}
```

- L'instruction **`if`** représente un **test**
- La paire d'accolade délimite un **bloc** de code

- Traduction: "Si la condition est vraie, alors exécute les instructions contenues dans le bloc de code"

if – forme et indentation

```
if (condition) {
  // instructions exécutées quand la condition est vraie
}
```

- L’accolade ouvrante se trouve sur la même ligne que la condition
- L’accolade fermante se trouve sur une nouvelle ligne
- Les instructions contenues dans le blocs sont indentées de 2 espaces

- **Il est indispensable de bien indenter son code pour qu’il soit lisible**

La notion de condition

- Une **condition** est une expression dont l'évaluation produit une valeur soit vraie, soit fausse : on parle de valeur **booléenne**

- Rappel: le type **Boolean** n’a que 2 valeurs possibles
  - **true** (vrai)
  - **false** (faux)

- Quand la valeur d'une condition est vraie, on dit que cette condition est **vérifiée**.

La notion de condition

```
if (true) {
  // la condition du if est toujours vraie :
  // les instructions de ce bloc seront toujours exécutées
}
if (false) {
  // la condition du if est toujours fausse :
  // les instructions de ce bloc seront jamais exécutées
}
```

La notion de condition

- Toute expression produisant une valeur booléenne (donc soit vraie, soit fausse) peut être utilisée comme condition dans une instruction **if**.

- Si la valeur de cette expression est **true**, le bloc de code associé au **if** sera exécuté

- On peut créer des expressions booléennes en utilisant les **opérateurs de comparaison**

Opérateur	Signification
==	Egale à
===	Egale à et même type
!=	Pas égale
!==	Pas égale ou pas même type
<	Plus petit
<=	Plus petit ou égale
>	Plus grand
>=	Plus grand ou égale

Attention

- La confusion entre
  - l'opérateur d'égalité **===** (ou **==**) et
  - l'opérateur d'affectation **=**
- Dans l'écriture d'une condition
  est une erreur très fréquente

Exprimer une alternative

```
Saisir un nombre
Si ce nombre est positif
  Afficher un message
Sinon
  Afficher un autre message
```

```
const nombre = Number(prompt("Entrez un nombre :"));
if (nombre > 0) {
  console.log(nombre + " est positif");
} else {
  console.log(nombre + " est négatif ou nul");
}
```

else

```
if (condition) {
  // instructions exécutées quand la condition est vraie
} else {
  // instructions exécutées quand la condition est fausse
}
```

- Une alternative s'exprime en JavaScript grâce à l'instruction **else** associée à un **if**
- Traduction: "Si la condition est vraie, alors exécute les instructions du bloc de code associé au **if**, sinon exécute celles du bloc de code associé au **else**"

Imbriquer des conditions

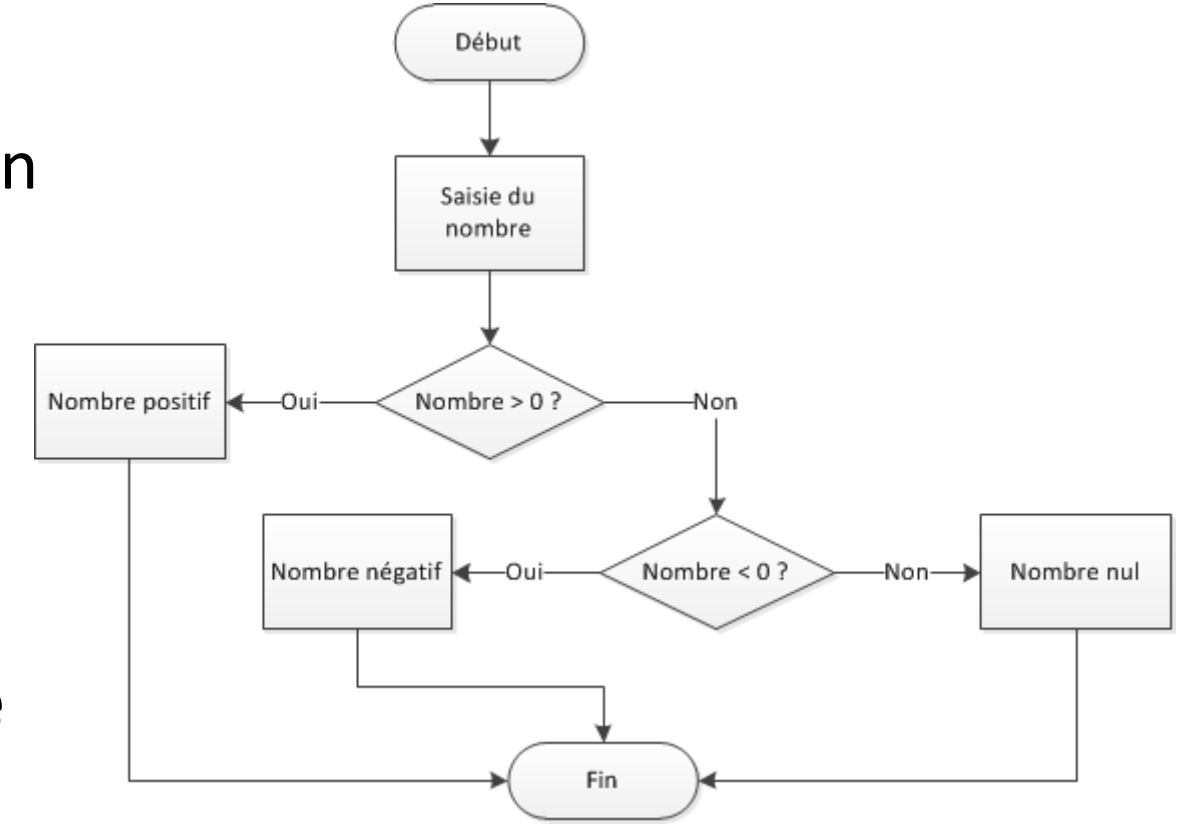
- Faites appel à votre sens logique pour combiner des conditions…

```
const nombre = Number(prompt("Entrez un nombre :"));
if (nombre > 0) {
  console.log(nombre + " est positif");
} else { // nombre <= 0
  if (nombre < 0) {
    console.log(nombre + " est négatif");
  } else { // nombre === 0
    console.log(nombre + " est nul");
  }
}
```

Imbriquer des conditions

- Il est possible de représenter graphiquement l'exécution du programme précédent au moyen d'un **diagramme de flux**

- Cet exemple nous montre que l'indentation permet de bien visualiser les différents blocs crées par les instructions **if/else**

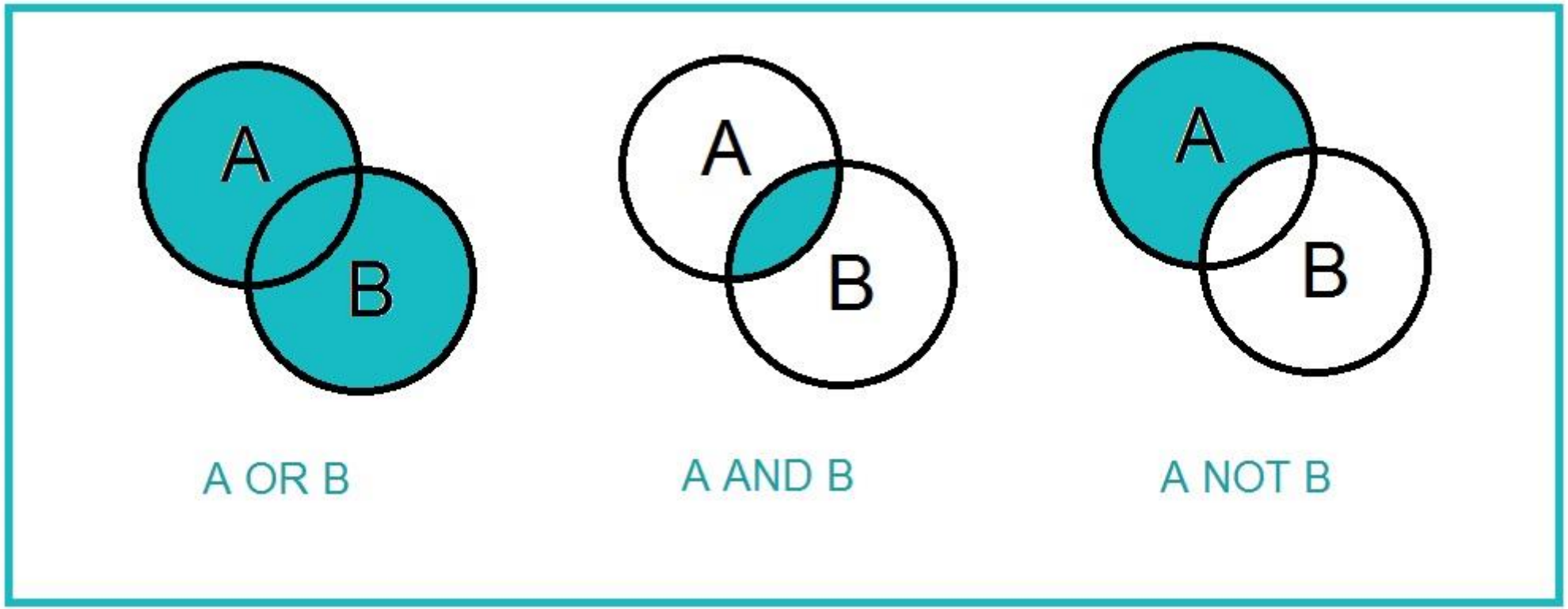


Imbriquer des conditions

- Si la seule instruction d'un bloc **else** est un **if**, il est possible d'écrire ce **if** sur la même ligne que le premier **else**, sans accolades ni indentation

```
const nombre = Number(prompt("Entrez un nombre :"));
if (nombre > 0) {
  console.log(nombre + " est positif");
} else if (nombre < 0) {
  console.log(nombre + " est négatif");
} else { // nombre === 0
  console.log(nombre + " est nul");
}
```

Créer des conditions composées



L‘opérateur logique ET

- La condition "nombre compris entre 0 et 100" peut s'exprimer sous la forme de deux sous-conditions
  - "nombre supérieur ou égal à 0"

  ET
  - "nombre inférieur ou égal à 100"

- Il faut que l'une **ET** l'autre de ces sous-conditions soient vérifiées.

```
if ((nombre >= 0) && (nombre <= 100)) {
  console.log(nombre + " est compris entre 0 et 100");
}
```

L‘opérateur logique ET

- L'opérateur **&&** (**ET** logique) s'applique à deux valeurs de type booléen.

- Son résultat est la valeur **true** uniquement si les deux valeurs auxquelles il s'applique valent **true**

```
console.log(true && true); // true
console.log(true && false); // false
console.log(false && true); // false
console.log(false && false); // false
```

L‘opérateur logique OU

- On souhaite maintenant vérifier qu'un nombre est en dehors de l'intervalle [0, 100].
- Pour satisfaire à cette condition, ce nombre doit
  - "être inférieur à 0"

  OU
  - " être supérieur à 100"

```
if ((nombre < 0) || (nombre > 100)) {
  console.log(nombre + " est en dehors de l'intervalle [0, 100]");
}
```

L‘opérateur logique OU

| => [AltGr] + [7]

- L'opérateur **||** (**OU** logique) s'applique à deux valeurs de type booléen.
- Son résultat est la valeur **true** si au moins une des deux valeurs auxquelles il s'applique vaut **true**.

- Voici la table de vérité de l'opérateur **||**.

```
console.log(true || true); // true
console.log(true || false); // true
console.log(false || true); // true
console.log(false || false); // false
```

L‘opérateur logique NON

- Il existe un troisième opérateur logique qui permet d'inverser la valeur d'une condition : l'opérateur NON.

- Il s'écrit en JavaScript sous la forme d'un point d'exclamation (!).

```
if (!(nombre > 100)) {
  console.log(nombre + " est inférieur ou égal à 100");
}
```

L‘opérateur logique NON

- L'opérateur **!** (NON logique) inverse une valeur de type booléen.

- Voici la table de vérité de cet opérateur

```
console.log(!true); // false
console.log(!false); // true
```

Exprimer un choix

Exprimer un choix

- Imaginons un programme qui conseille l'utilisateur sur la tenue à porter en fonction de la météo actuelle.

- Une première solution consiste à utiliser des instructions **if/else**

Exprimer un choix

```
const meteo = prompt("Quel temps fait-il dehors ?");
if (meteo === "soleil") {
  console.log("Sortez en t-shirt.");
} else if (meteo === "vent") {
  console.log("Sortez en pull.");
} else if (meteo === "pluie") {
  console.log("Sortez en blouson.");
} else if (meteo === "neige") {
  console.log("Restez au chaud à la maison.");
} else {
  console.log("Je n'ai pas compris.");
}
```

Exprimer un choix

- Lorsqu'un programme consiste à déclencher un bloc d'opérations parmi plusieurs selon la valeur d'une expression, on peut l'écrire en utilisant l'instruction **switch**

```
const meteo = prompt("Quel temps fait-il dehors ?");
switch (meteo) {
  case "soleil":
    console.log("Sortez en t-shirt.");
    break;
  case "vent":
    console.log("Sortez en pull.");
    break;
  case "pluie":
    console.log("Sortez en blouson.");
    break;
  case "neige":
    console.log("Restez au chaud à la maison.");
    break;
  default:
    console.log("Je n'ai pas compris.");
}
```

switch

- L'instruction **switch** déclenche l'exécution d'un bloc d'instructions parmi plusieurs possibles. Seul le bloc correspondant à la valeur de l'expression testée sera pris en compte

```
switch (expression) {
  case valeur1:
    // instructions exécutées quand expression vaut valeur1
    break;
  case valeur2:
  // instructions exécutées quand expression vaut valeur2
  default:
  // instructions exécutées quand aucune des valeurs ne correspond
}
```

switch

- Il n'y a pas de limite au nombre de cas possibles.
- Le mot-clé **default**, à placer en fin de **switch**, est optionnel. Il sert souvent à gérer les cas d'erreurs.
- Si on omet de mettre l’instruction **break** le programme va commencer d’exécuter le bloc remplissant la condition puis continuera à exécuter les blocs suivant jusqu’à la fin ou jusqu’au prochain **break**.

switch

- On peut aussi assembler les valeurs des « case ».

```
switch (expression) {
  case valeur1:
  case valeur2:
    // instructions exécutées quand expression vaut valeur1 ou valeur2
    break;
  default:
  // instructions exécutées quand aucune des valeurs ne correspond
}
```

Exercices

- Exercices 4.x : **Conditions**